CISM higher-order dynamics: model solution

From Interactive System for Ice sheet Simulation

Contents

Governing Equations

The final form of the equations we'd like to solve is:

$$x: \quad 4\frac{\partial}{\partial x}\left(\eta\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(\eta\frac{\partial u}{\partial y}\right)+\frac{\partial}{\partial z}\left(\eta\frac{\partial u}{\partial z}\right)=-2\frac{\partial}{\partial x}\left(\eta\frac{\partial v}{\partial y}\right)-\frac{\partial}{\partial y}\left(\eta\frac{\partial v}{\partial x}\right)+\rho g\frac{\partial s}{\partial x}$$

$$y: \quad 4\frac{\partial}{\partial y}\left(\eta\frac{\partial v}{\partial y}\right)+\frac{\partial}{\partial x}\left(\eta\frac{\partial v}{\partial x}\right)+\frac{\partial}{\partial z}\left(\eta\frac{\partial v}{\partial z}\right)=-2\frac{\partial}{\partial y}\left(\eta\frac{\partial u}{\partial x}\right)-\frac{\partial}{\partial x}\left(\eta\frac{\partial u}{\partial y}\right)+\rho g\frac{\partial s}{\partial y}$$

Coordinate Transform

For ice sheet modeling, it is convenient to recast the governing equations using a dimensionless, stretched vertical coordinate (often called a *sigma coordinates* (http://atmo.tamu.edu/class/metr452/models/2001/vertres.html)). The stretched vertical coordinate is defined as:

$$\sigma = \frac{(s-z)}{H}$$

This means that at the surface of the ice sheet $\sigma = 0$, and at the base $\sigma = 1$ regardless of the ice thickness. As a result of this transformation, a coordinate (x,y,z) is mapped to (x',y',σ). This means that function derivatives must be re-written (using $\frac{\partial f}{\partial x}$ as an example) as:

$$\frac{\partial f}{\partial x}=\frac{\partial f}{\partial x'}\frac{\partial x'}{\partial x}+\frac{\partial f}{\partial y'}\frac{\partial y'}{\partial x}+\frac{\partial f}{\partial \sigma}\frac{\partial \sigma}{\partial x}$$

Similarly for $\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial z}$. We can simplify this by assuming that

$$\frac{\partial x'}{\partial x}, \frac{\partial y'}{\partial y} = 1$$

and

$$\frac{\partial x'}{\partial y}, \frac{\partial x'}{\partial z}, \frac{\partial y'}{\partial x}, \frac{\partial y'}{\partial z} = 0.$$

This assumption is valid if the bed and surface gradients are not too large. This simplifies the above to:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial x'} + \frac{\partial f}{\partial \sigma}\frac{\partial \sigma}{\partial x}$$

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial y'} + \frac{\partial f}{\partial \sigma}\frac{\partial \sigma}{\partial y}$$

$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial \sigma}\frac{\partial \sigma}{\partial z}$$

Rescaling parameters a_x, a_y, b_x, b_y, and c_{xy} are defined. For the *x* derivative case (the *y* derivative case is analogous) we have

$$a_x = \frac{1}{H}(\frac{\partial s}{\partial x'} - \sigma\frac{\partial H}{\partial x'})$$

$$b_x = \frac{\partial a_x}{\partial x'} + a_x\frac{\partial a_x}{\partial \sigma} = \frac{1}{H}(\frac{\partial^2 s}{\partial x'^2} - \sigma\frac{\partial^2 H}{\partial x'^2} - 2a_x\frac{\partial H}{\partial x'})$$

$$c_{xy} = \frac{\partial a_y}{\partial x'} + a_x\frac{\partial a_y}{\partial \sigma} = \frac{\partial a_x}{\partial y'} + a_y\frac{\partial a_x}{\partial \sigma}$$

Using these, expressions for the *x* derivatives become:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial x'} + a_x\frac{\partial f}{\partial \sigma}$$

$$\frac{\partial}{\partial x}\left(\eta\frac{\partial u}{\partial x}\right) = \frac{\partial}{\partial \hat{x}}\left(\eta\frac{\partial u}{\partial \hat{x}}\right) + \frac{\partial \sigma}{\partial \hat{x}}\frac{\partial}{\partial \sigma}\left(\eta\frac{\partial u}{\partial \hat{x}}\right) + \frac{\partial \sigma}{\partial \hat{x}}\frac{\partial}{\partial \hat{x}}\left(\eta\frac{\partial u}{\partial \sigma}\right) + \left(\frac{\partial \sigma}{\partial \hat{x}}\right)^2\frac{\partial}{\partial \sigma}\left(\eta\frac{\partial u}{\partial \sigma}\right) + \left(\frac{\partial^2 \sigma}{\partial \hat{x}^2}\right)\eta\frac{\partial u}{\partial \sigma}$$

where hatted values refer to the coordinate directions in sigma coordinates. Similarly, the first cross-stress term on the RHS is given by

$$\frac{\partial}{\partial x}\left(\eta\frac{\partial v}{\partial y}\right) = \frac{\partial}{\partial \hat{x}}\left(\eta\frac{\partial v}{\partial \hat{y}}\right) + \frac{\partial \sigma}{\partial \hat{x}}\frac{\partial}{\partial \sigma}\left(\eta\frac{\partial v}{\partial \hat{y}}\right) + \frac{\partial \sigma}{\partial \hat{y}}\frac{\partial}{\partial \hat{x}}\left(\eta\frac{\partial v}{\partial \sigma}\right) + \frac{\partial \sigma}{\partial \hat{x}}\frac{\partial \sigma}{\partial \hat{y}}\frac{\partial}{\partial \sigma}\left(\eta\frac{\partial v}{\partial \sigma}\right) + \frac{\partial^2 \sigma}{\partial \hat{x}\partial \hat{y}}\eta\frac{\partial v}{\partial \sigma}$$

One term has become five terms and each one of those is pretty ugly looking on its own. Luckily, there is a lot of

symmetry here. Notice that if we wanted to design subroutines to discretize the terms on the RHS, we could re-use a lot of them by either applying them to the correct velocity component (to either the *U* or the *V* discretization) or by passing the appropriate arguments (by passing either the grid spacing in the *X* direction or the *Y* direction, where appropriate).

A similar transform is applied to each of the terms in the governing equations given above. At any point within the grid, the grid spacing, coordinate transform, and viscosity information associated with the unknown velocity components (*U* and *V*) is made discrete using finite differences. This information ultimately equates to coefficients on the unknown velocities, allowing the governing equations over the entire grid (with appropriate discretizations for boundary conditions) to be recast as a system of *n* equations in *n* unknowns. In turn, this system is solved using standard linear algebraic methods for large, [sparse (http://en.wikipedia.org/wiki/Sparse_matrix)] systems of equations.

Operating Splitting

In the governing equations given above, note that for the *x* equation we have moved all terms containing gradients in *v* to the right-hand side (RHS) (and vice-versa for the *y* equation).

This allows us to solve the equations using an *operator splitting* approach; for the *x* equation, we treat *v* as known (where we take the values of *v* from the previous iteration, as discussed further below) and solve for *u*, and vice versa when we solve the *y* equation for *v*. The "splitting" refers to the fact that we are breaking the multi-dimensional divergence operation into two steps; rather than solving one big matrix equation for *u* and *v* simultaneously, we solve two smaller matrix equations in sequence with one of the unknowns treated as a known "source" term. This procedure was probably more common and important years ago when it was desirable to keep the matrix equations as small as possible for memory management issues. On today's machines, with fewer memory limitations (in particular when dealing with codes designed to run on parallel, distributed memory architectures) this splitting is not necessary and may even lead to some undesirable numerical side effects (i.e. a slow-down in the convergence of iterations used to treat nonlinearity in the governing equations).

A general matrix form of the "split" equations, where coefficients on the *u* and *v* velocity components (i.e. viscosity, grid spacing, scalars) are contained in the block matrices **A**, is given by

$$\begin{bmatrix} \mathbf{A_{uu}} & \mathbf{0} \\ \mathbf{0} & \mathbf{A_{vv}} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{b_u} - \mathbf{A_{uv}v} \\ \mathbf{b_v} - \mathbf{A_{vu}u} \end{bmatrix}$$

$$\mathbf{A_{uu}u} = \mathbf{b_u} - \mathbf{A_{uv}v}, \qquad \mathbf{A_{vv}v} = \mathbf{b_v} - \mathbf{A_{vu}u}$$

where the **uu** subscript denotes block matrices containing coefficients for gradients on *u* in the equation for the *x* component of velocity (i.e. *u*). The subscript **uv** denotes block matrices containing coefficients for gradients on *v* in the equation for the *x* component of velocity (and similarly for the **vv** and **vu** subscripts). On the right-hand side, the single subscripts **u** and **v** are attached to the geometric source terms for the *x* and *y* components of velocity, respectively.

Solution of the Non-linear System Through a Fixed Point Iteration

The non-linearity in the equations - the fact that the coefficients on the velocity components (the viscosity) are dependent on the velocity (or more specifically, the velocity gradients) - is handled through a "fixed-point iteration" (http://en.wikipedia.org/wiki/Fixed_point_iteration) . A general fixed point iteration for a vector of unknowns *u* can be written as

$$u^k = \mathbf{B}\left(u^{k-1}\right),$$

where *k* is the index for the *u* being solved for and **B** is a matrix operation performed on the components of *u* obtained at the previous iteration, *k-1*. The "fixed point" occurs when the values of *u* at *k* and *k-1* are equal to within some given tolerance (at which point the iteration process is halted). CISM has options for implementing both "Picard" and "Newton"-based fixed-point iterations. For the Picard iteration (standard in CISM), the matrix coefficients with a velocity dependence are simply based on the velocities obtained at the previous iteration. In most cases, this equates to using velocities obtained from the previous iteration to calculate the strain rate components that go into the calculation of the effective viscosity, η.

Final Matrix Form

When accounting for both the operator splitting and the Picard iteration on the effective viscosity, the final form of the matrix equations solved in CISM becomes

$$\begin{bmatrix} \mathbf{A}_{\mathbf{uu}}^{k-1} & 0 \\ 0 & \mathbf{A}_{\mathbf{vv}}^{k-1} \end{bmatrix} \begin{bmatrix} \mathbf{u}^{k} \\ \mathbf{v}^{k} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_{\mathbf{u}} - \mathbf{A}_{\mathbf{uv}}^{k-1}\mathbf{v}^{k-1} \\ \mathbf{b}_{\mathbf{v}} - \mathbf{A}_{\mathbf{vu}}^{k-1}\mathbf{u}^{k-1} \end{bmatrix}$$

$$\mathbf{A}_{\mathbf{uu}}^{k-1}\mathbf{u} = \mathbf{b}_{\mathbf{u}} - \mathbf{A}_{\mathbf{uv}}^{k-1}\mathbf{v}^{k-1}, \qquad \mathbf{A}_{\mathbf{vv}}^{k-1}\mathbf{v} = \mathbf{b}_{\mathbf{v}} - \mathbf{A}_{\mathbf{vu}}^{k-1}\mathbf{u}^{k-1}$$

where the index *k* denotes an unknown value being solved for during the current non-linear iteration and the index *k*-1 denotes a lagged value taken from solution at the end of the previous non-linear iteration (again, here the lagging is primarily with respect to the effective viscosity, the value of which is calculated using velocity gradients obtained at the end of the previous iteration). The final form of the matrix equations given above represents a linear system; for the solution at any particular nonlinear iteration *k*, all of the coefficients on the unknown velocity components *u* and *v* are held "frozen" during the solution of the linear system. This linear system can be solved using any practical method. For large, sparse systems, some variant on the iterative conjugate gradient method (e.g. BiCG, GMRES) is generally the most efficient. In this case the linear system is not solved exactly but is solved to within some small tolerance of the "true" solution.

Newton-based Methods for Solutions of the Non-linear System

Without any operator splitting, the generic matrix form of the equations to be solved can be written as

$$\mathbf{A}(\mathbf{u})\mathbf{u} = \mathbf{b}.$$

The linearized form of the equations to be solved using the Picard solution can be written as

$$\mathbf{u}^{k} = \mathbf{A}(\mathbf{u}^{k-1})^{-1}\mathbf{b}.$$

The full nonlinear system to be solved can be written as

$$\mathbf{F}(\mathbf{u}) = \mathbf{A}(\mathbf{u})\mathbf{u} - \mathbf{b}$$

with the solution for the uknown vector *u* given by

$$\mathbf{F}(\mathbf{u}) = 0.$$

A Newton-based solution for this system of equations, based on a first-order Taylor series expansion about the solution for *u* at iteration *k-1*, can be written as

$$\mathbf{F}(\mathbf{u}^k) = \mathbf{F}(\mathbf{u}^{k-1}) + \mathbf{F}'(\mathbf{u}^{k-1})\delta\mathbf{u}^{k-1},$$

where

$$\mathbf{F}'(\mathbf{u}^{k-1}) = \mathbf{J}^{k-1}$$

is the system Jacobian with individual components give by

$$J_{ij} = \frac{\partial F_i(\mathbf{u}^{k-1})}{\partial u_j}$$

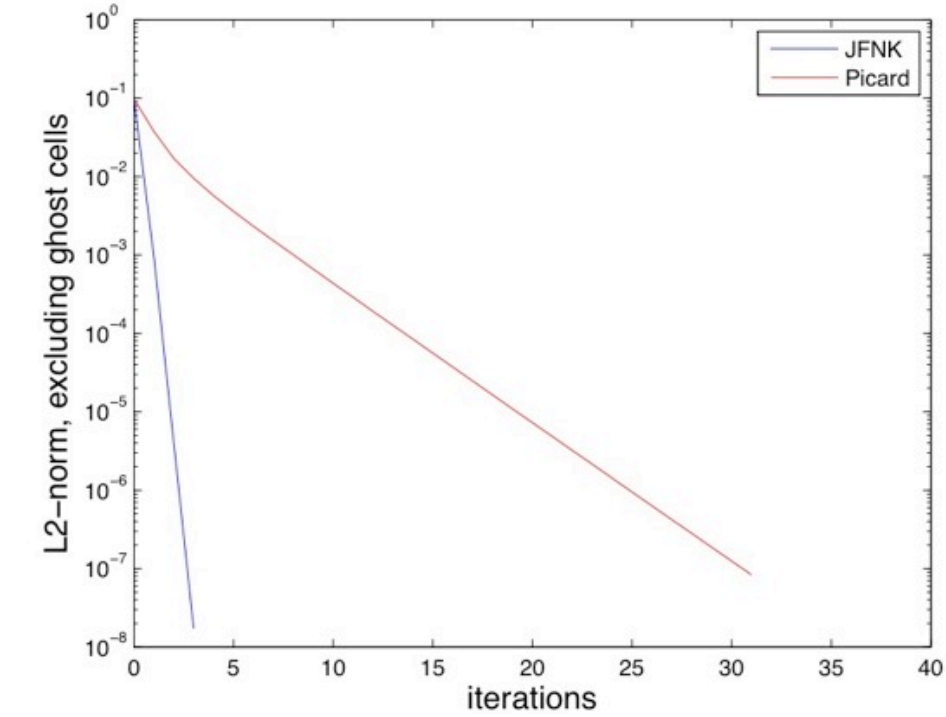
and

$$\delta\mathbf{u}^{k-1} = \mathbf{u}^k - \mathbf{u}^{k-1}$$

is the Newton update to be solved for. One method for doing so is by solving

$$\delta\mathbf{u}^{k-1} = -\left(\mathbf{J}^{k-1}\right)^{-1}\mathbf{F}(\mathbf{u}^{k-1}).$$

The advantage of Newton-based methods is that, with a good initial guess for the solution, convergence rates are very often quadratic (e.g. the residual decreases quadratically, so that at iteration *k* one has a residual of 0.1, at iteration *k+1*, a residual of 0.01, and at iteration *k+2* a residual of 0.0001), whereas Picard-based iterations are much slower to converge. A figure comparing rates of convergence for Picard versus Newton on a CISM tests case is shown below. The Newton method is based on the work of Lemieux et al. (submitted to *JCP*), which is discussed further below.



Rates of convergence for the nonlinear iteration in CISM.

The Jacobian Free Approach

In practice, the model Jacobian may either be too difficult or to expensive too form. A "Jacobian Free Newton-Krylov" (JFNK) approach has recently been implemented in CISM (Leimieux et al., submitted to *JCP*), largely following methods discussed in Knoll and Keyes (2004) (http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6WHY-49KSNJ6-4&_user=2493154&_coverDate=01%2F20%2F2004&_alid=1678413922&_rdoc=1&_fmt=high&_orig=search&_origin=search&_zone=rslt_list_item&_cdi=6863&_sort=r&_st=13&_docanchor=&view=c&_ct=1&_acct=C000057551&_version=1&_urlVersion=0&_userid=2493154&md5=887f929a8adbac698ecf277ac0c99cef&searchtype=a) . The crux of the method comes from noting that, when solving the last equation above using a Krylov method (e.g. Conjugate Gradients, GMRES, etc.) the solution for the Newton update is taken from a combination of Krylov vectors that span the subspace

$$span\left\{\mathbf{r}_0, \mathbf{J}\mathbf{r}_0, \mathbf{J}^2\mathbf{r}_0, ..., \mathbf{J}^{n-1}\mathbf{r}_0\right\} = span\left\{\mathbf{r}_0, \mathbf{J}\mathbf{v}_1, \mathbf{J}\mathbf{v}_2, ..., \mathbf{J}\mathbf{v}_{n-1}\right\}.$$

This implies that, when using a Krylov method, one only ever needs to calculate matrix vector products of the form $\mathbf{J}\mathbf{v}$ when building up the subspace that approximates the solution vector $\delta\mathbf{u}$.

Following Knoll and Keyes (2004), note that the necessary matrix vector products can be approximated through nonlinear function evaluations and a perturbation as

$$\mathbf{J}\mathbf{v} \approx \frac{\mathbf{F}\left(\mathbf{u} + \varepsilon\mathbf{v}\right) - \mathbf{F}\left(\mathbf{u}\right)}{\varepsilon}.$$

It is not immediately obvious why this approximation is valid. To verify this, take a few steps back and consider a nonlinear system of equations of two variables, *u1* and *u2*. The right-hand side of the above equation can be expanded as

$$\frac{\mathbf{F}\left(\mathbf{u}+\varepsilon\mathbf{v}\right)-\mathbf{F}\left(\mathbf{u}\right)}{\varepsilon}=\begin{bmatrix}\frac{F_1(u_1+\varepsilon v_1,u_2+\varepsilon v_2)-F_1(u_1,u_2)}{\varepsilon}\\ \frac{F_2(u_1+\varepsilon v_1,u_2+\varepsilon v_2)-F_2(u_1,u_2)}{\varepsilon}\end{bmatrix}.$$

A first-order Taylor series expansion approximation to this is given by

$$\frac{\mathbf{F}\left(\mathbf{u}+\varepsilon\mathbf{v}\right)-\mathbf{F}\left(\mathbf{u}\right)}{\varepsilon}\approx\begin{bmatrix}\frac{F_1(u_1,u_2)+\varepsilon v_1\frac{\partial F_1}{\partial u_1}+\varepsilon v_2\frac{\partial F_1}{\partial u_2}-F_1(u_1,u_2)}{\varepsilon}\\ \frac{F_2(u_1,u_2)+\varepsilon v_1\frac{\partial F_2}{\partial u_1}+\varepsilon v_2\frac{\partial F_2}{\partial u_2}-F_2(u_1,u_2)}{\varepsilon}\end{bmatrix},$$

which collapses to

$$\frac{\mathbf{F}\left(\mathbf{u}+\varepsilon\mathbf{v}\right)-\mathbf{F}\left(\mathbf{u}\right)}{\varepsilon}\approx\begin{bmatrix}v_1\frac{\partial F_1}{\partial u_1}+v_2\frac{\partial F_1}{\partial u_2}\\ v_1\frac{\partial F_2}{\partial u_1}+v_2\frac{\partial F_2}{\partial u_2}\end{bmatrix}.$$

Finally, note that the right-hand side of the above equation is equal to

$$\mathbf{J}\mathbf{v}\approx\begin{bmatrix}v_1\frac{\partial F_1}{\partial u_1}+v_2\frac{\partial F_1}{\partial u_2}\\ v_1\frac{\partial F_2}{\partial u_1}+v_2\frac{\partial F_2}{\partial u_2}\end{bmatrix},$$

with the Jacobian matrix given by

$$\mathbf{J}=\begin{bmatrix}\frac{\partial F_1}{\partial u_1} & \frac{\partial F_1}{\partial u_2}\\ \frac{\partial F_2}{\partial u_1} & \frac{\partial F_2}{\partial u_2}\end{bmatrix}.$$

This matrix vector product is what needs to be calculated repeatedly while building up the Krylov subspace vectors that combine to approximate the Newton update vector $\delta\mathbf{u}$. The important point is that at no point in this process does one need to calculate the entire Jacobian matrix. Another important point is that the accuracy of the approximation to the Jacobian is proportional to the small perturbation term, ε.

Go to Blatter-Pattyn model.

Go to Blatter-Pattyn Boundary Conditions.

Go to CISM governing equations and numerical solution.

Return to main coarse page